Bachelor's Thesis

Classifying Core Developers in Open-Source Software Projects

A Responsibility Driven Approach

Philipp Scholtes

March 21, 2024

Advisor:
Christian Hechtl Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering
Prof. Dr. Jilles Vreeken CISPA

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Most Open-Source Software (OSS) projects are developed by hundreds of developers acting as a community. Most studies that investigated these communities have classified developers into a core and a peripheral group and conducted analyses.
However, the developers in their groups were mostly not investigated based on their specific type of contribution to the software project. This information can improve developers' and researchers' understanding of effective collaboration among developers specific to their respective project areas.
In this thesis, we construct two types of developer networks, using cochange data and issue data and apply the eigenvector and hierarchy centrality metrics to classify the developers into core and peripheral. We use these classifications to create three types of core developers. These include developers who are core in both areas (cochange and issue) and those who are core in only one area. We investigate each group by their characteristics and centrality score. Our results indicate, that core developers in both areas rank higher in the two developer networks regarding their centrality and have more commits and structural changes than the other core developer groups. Additionally, core developers of both areas also participate in issues in a more important function, e.g., approving a new version.
Furthermore, we take a closer look at core developers in both areas. We use two indices, one to evaluate the collaborations and one to evaluate the interactions, to identify substantial differences between the core developers and find five subgroups with different characteristics and activity patterns, allowing us to differentiate core developers in a more nuanced way. Using these five subgroups, we analyze their distribution and movement between them, as well as their entry and exit into the core developer group over different time periods, to find many specific transition patterns between each subgroup. Newcomers and returners mostly tend to join in community-driven tasks or with a wider spectrum of community-driven and influential contribution tasks. Those core developers who leave the core developer group mostly leave the core developer group completely.

Contents

List of Figures

List of Tables

Listings

Acronyms

OSS Open-Source Software

PR pull request

1 Introduction

In today's software development, OSS projects become more and more important and therefore are interesting as a subject for different research areas.

The developer community in an OSS project often includes multiple hundreds of developers which makes the manual analysis of such a community difficult. A common approach of studies is to analyze these communities by classifying the developers into the core and peripheral roles [4, 5, 7, 8, 12, 28]. Other studies have focused on the core developers themselves without looking into the different types of core developers that are in an OSS project. Developers do not always contribute to the code and participate actively in issues but rather do just one of them, but very intense. We classify the developers into the core and peripheral roles by using code commits and participation in issues, namely cochange data and issue data, for our network-based approach, because it has proven to be a reliable approach with additional insights into the communities [12]. For the classification metric, we use the eigenvector centrality as it also proves to capture structural differences in developer relationships and has similar performance in detecting core developers and precision as the other classification metrics [4]. In addition, we apply the hierarchy centrality as classification metric, as it gives an insight into the community structure of a developer network [21].

We can use developer networks to create those different groups of core developers in an OSS project to give an insight into their characteristics and activity patterns. In addition, we analyze the transitions within these subgroups, as well as the entry and exit of core developers over time to find a more nuanced understanding of core developer roles. Within each subgroup, we find many specific patterns of transition. Newcomers and returnees tend to join with community-oriented tasks, or with a wider range of community-oriented and influential contributing tasks. Most core developers who leave the core developer group tend to leave the core developer group completely. This information can be used for future research to improve developers' and researchers' understanding of effective collaboration among developers specific to their respective project areas.

1.1 Goal of this Thesis

In this thesis, we want to investigate the characteristics and activity patterns of different types of core developers in OSS projects to find a more nuanced understanding of core developer roles. The two areas we investigate are the cochange data and the issue data

of an OSS project in the cloud-based hosting service GitHub[1]. With the data we collect from Git[2], a version-control system, we build developer networks of cochange data and the issue data which we use to classify the developers into the core and peripheral roles (Section 2.2) by using the eigenvector centrality and hierarchy centrality metrics. This gives us the core developers of technical and social core developers which we will intersect to get the core developers in both areas. We then sperate those into three different groups of core developers, the developers who are core in the technical and the social area and those who are just core in only one area. With this separation, we can identify potential subgroups within the core developer group to help us understand their responsibilities and tasks in an OSS project. In addition, we want to find patterns in the transitions over time within the subgroups, as well as patterns in those developers who become core developers and those who leave the core developer group.

1.2 Overview

In Chapter 2, we explain what OSS is and give a deeper insight into OSS projects. We also explain developer roles, what they are, and how we classify them in our research. Lastly, we provide information about the network theory, how we build our networks and what data we use to build them, what developer networks are, and what eigenvector centrality and hierarchy centrality are, which we use to classify our developers.

In Chapter 3, we present our research questions and methodology. This includes which projects we use in our research, how we build our networks, our analysis of these networks, and the operationalization of two indices to separate core developers into different groups with distinct activity patterns as well as the time-based approach of these indices.

In Chapter 4, we present our results of the analyses, how we separate core developers into the groups we find, the results of those findings, we discuss our results, and we discuss the internal and external threats to the validity of our results.

In Chapter 5, we present the related work that is related to our thesis topic.

In Chapter 6, we give the conclusion of this thesis and provide an outlook on future work.

1 https://github.com/
2 https://git-scm.com/

2

Background

In this chapter, we explain the necessary background information to understand the presented work. First, we will introduce the concept of OSS development. Then we will explain what developers' roles are, and go into detail about the core developer role. Lastly, we explain network analysis, therefore we explain networks, network building, which data we use for our network building, developer networks, and eigenvector and hierarchy centrality which we use for our classification of developers in the network.

2.1 Open-Source Software

OSS stands for open-source software and refers to a specific type of software development and licensing model. OSS is freely available, meaning that its source code is freely available to the public to view, use, modify, and redistribute as long as it complies with the license terms. There are many OSS projects that are developed and maintained by many developers who contribute from all over the world. These developers work together as a community to improve the software. Because the code is open and developers work in a community, fixing problems and innovating, OSS is very transparent. There are a wide variety of projects, from operating systems (e.g. Linux) to application software (e.g. Mozilla Firefox) and others. As a result, OSS offers possible benefits such as lower costs, flexibility, security, and independence from software vendors[2, 9].

2.2 Developer Roles

In the context of networks, OSS, and software development in general, developer roles refer to the various responsibilities and tasks that individuals take on within a software project. Nakakoji et al. [18] have introduced the onion model, which includes eight different developer roles, three for project users and five for code contributors. A different approach to the developer roles has been taken by Xu et al. [27], who introduced four developer roles. These four or five developer classifications approaches are very fine-grained and the most common approach is to divide the developers into just two groups, the core developers and the peripheral developers [7, 12, 25], which is what we will also use in this thesis.
The core developers of a software project have a major impact on the software project, as they are taking maintenance tasks, implement core functionality, and have deep knowledge of the project or specific subsystems. Peripheral developers, though not part of the core team, still make valuable contributions to a software project. They handle tasks like addressing

minor bugs, providing occasional enhancements, and participating in discussions or code reviews [4, 7, 12, 19].

2.3 Network Analysis

Networks are concepts used to study relationships between people in various activities. Networks find widespread application in various research domains, they are built and analyzed to gain deeper insight into the developer's activities and the connection between the people in the network. Throughout this thesis, we utilize networks to analyze interactions between developers within OSS projects as well as contributions of developers to the same file in an OSS project [11, 26].
In the following sections, we give a deeper insight into the mathematical definition of a network in Section 2.3.1. Further details on developer networks are presented in Section 2.3.2. Additionally, in Section 2.3.3, we provide essential insights into the classification metrics we use, the eigenvector centrality and the hierarchy centrality, which are necessary for this thesis.

2.3.1 Network Theory

Networks are represented using graphs, defined as a tuple $G = (V, E)$, where V represents a set of vertices and E represents a set of edges. An edge $e \in E$ links two vertices, meaning the set of edges can be expressed as $E = \{(u, v) | u, v \in E\}$. Graphs can be directed, which means edges have a specific direction, or undirected, therefore having bidirectional edges. Formally, in a directed graph, $(u, v) \neq (v, u)$, where (u, v) is an edge from $u \in V$ to $v \in V$. In an undirected graph, an edge from u to v is identical to an edge from v to u, meaning $(u, v) = (v, u)$. Graphs can be weighted, where each edge has a weight of a natural number, defined by the function $\omega : E \to \mathbb{N}$, which assigns weights to edges. Unweighted graphs have $\omega = 1$ for all $e \in E$. Graphs can also be simplified by converting all edges, which connect the same vertices in the same direction, into one edge. The weight of a simplified edge is the sum of the weights of the edges in the original graph. If the original graph is unweighted, the simplified version retains the same information, as the weight of an edge in the simplified graph corresponds to the number of edges connecting the same vertices in the original graph[24, 26].

2.3.2 Developer Networks

Developer networks indicate the connection between developers of a project by a specific type of interaction meaning that the developers are represented as the vertices of the network. In this thesis, we use data from Git repositories to build developer networks, which is a common approach in software engineering, particularly for analyzing OSS projects [1, 4, 5, 14, 25]. We build developer networks of an OSS project by using the cochange (Section 2.3.2.1) and the issue (Section 2.3.2.2) data.

2.3.2.1 *Cochange Data*

Developer collaboration networks are developer networks that are built from cochange (commit) data from a version-control system. Developers in this network are connected if they have contributed to the same file or function in the project [11, 15]. We will only consider undirected edges between developers, because we are not interested in how they contribute to the code (who commits first or makes changes to the other ones' commit), but are only interested in if they are connected to each other. We are simplifying the complexity of the network as it aids in faster computation without altering the results. We can do this, as Bock et al. [4] have shown, that the differences are marginal in OSS projects, because we only want to classify the core developers in this network.

2.3.2.2 *Issue Data*

Developer communication networks are developer networks that are built from communication data. The communication data is normally taken from the communication channel of the project which can include the mailing list or the issues. For our study, we will take the issue data of the OSS project to build the developer communication network. Developers in this network are connected if they have interacted or participated in the same issue [11]. We will only consider undirected edges between developers, because we are not interested in how they participated in issues, e.g. just commenting or closing pull requests (pull request (PR)), but we are only interested in whether they are connected to each other. We are simplifying the complexity of the network, regarding the weakness that we do not know how they participated in issues, but we are foremost interested in the classification into the core and peripheral roles. We can do this, as Bock et al. [4] have shown, that the differences are marginal in OSS projects.

2.3.3 Eigenvector Centrality

Eigenvector centrality is a metric for measuring the influence of a node within a network, that extends the concept of degree centrality [23]. While degree centrality simply counts the total number of connected nodes to determine a node's centrality, eigenvector centrality takes into account not only the number of neighboring nodes but also the importance of those neighboring nodes. In eigenvector centrality, connections to influential individuals carry more weight, meaning that being connected to an individual with multiple connections gives an individual a higher value than being connected to less connected individuals. Eigenvector centrality is calculated by evaluating how well an individual is connected to other highly influential parts of the network [6].
We will use this metric in our analysis as Bihari and Pandia [3] suggest, because it is better at finding core authors in relationship networks than other centrality classifications.

2.3.4 Hierarchy Centrality

Hierarchy centrality is a metric used to measure the community structure of a developer network. Hierarchy centrality takes into account the number of connections to an individual (node degree) and the density of local connections (clustering coefficient). In hierarchy centrality, an individual is identified as a core developer if they have a high node degree and a low clustering coefficient, resulting in a high hierarchy centrality. A lower node degree and a higher clustering coefficient indicate a lower hierarchy centrality and define a peripheral developer. This means that a developer has a high hierarchy centrality by being connected to many developers who are loosely connected to each other. A lower hierarchy centrality indicates a developer being connected to fewer developers with a tighter connection to each other. Hierarchy centrality is calculated by dividing the node degree by the clustering coefficient [4, 12, 13, 21].

3

Methodology

In this chapter, we present our research question as well as an overview of the OSS projects we analyze in this thesis, our approach, and our implementation to build and analyze networks. Finally, we will provide our expectations for this research and the internal and external threats to validity.

3.1 Research Questions

In this section, we present our research question. The goal of our research is to get a better understanding of the responsibilities of core developers, which we identify as such through analysis of the developer networks based on the cochange and issue data.

RQ1: *What specific characteristics or patterns of activity distinguish core developers who play a central role in both the cochange data and issue data of the OSS project from those who dominate in only one of these areas?*
RQ1 concentrates on the unique characteristics and activities of these two groups. Our goal is to compare these traits to better understand how these core developers differ from each other.

RQ2: *How do the activity patterns of core developers in OSS projects evolve over time?*
RQ2 is to shed light on the activity patterns of core developer and how they evolve over time. We aim to shed light on a more nuanced understanding on core developer roles and their dynamics.

With these research questions, we aim to analyze the characteristics and patterns of activity among core developers who are identified as core developers in both cochange data and issue data. In addition, our goal is to analyse these core developers and their activity patterns over time in detail.
This research could provide insights into the roles and responsibilities of core developers in these groups and their potential influence on the project.

3.2 Projects

In this section, we present the projects we analyze in this thesis, all OSS projects on GitHub. We have chosen a sample of ten different OSS projects for our research. All projects have been active for multiple years and have multiple hundreds of developers which can be separated into the different core developer groups. The exact numbers of commits are listed in Table 3.1 and information for the extracted numbers of issue comments can be found in Table 3.2. We can already see, that the number of developers, who commit changes, is significantly lower than the number of participants in issues as well as the number of commits is lower than the number of issue comments.

Table 3.1: Number of commits, the number of recognized core developers via eigenvector and hierarchy centrality, and the observation period the commits are extracted from for each project

Project	# Commits	# Core Developers (Eigenvector)	# Core Developers (Hierarchy)	Observation period
Angular	18.869	136	136	2014-09-18 - 2020-09-25
Atom	33.382	60	60	2011-08-19 - 2020-12-10
Bootstrap	16.249	44	44	2011-04-27 - 2020-12-22
Deno	4.913	70	70	2018-05-14 - 2020-12-22
Moby	22.845	232	232	2013-01-19 - 2020-12-22
OpenSSL	25.438	84	84	1998-12-21 - 2020-02-17
React	11.183	160	160	2013-06-03 - 2020-12-22
Reveal.js	2.248	28	28	2011-06-07 - 2020-10-12
Tensorflow	92.450	304	304	2015-11-07 - 2020-12-22
VSCode	68.357	201	201	2015-11-13 - 2020-12-22

Table 3.2: Number of issues, the number of recognized core developers via eigenvector and hierarchy centrality, and the observation period the issues are extracted from for each project

Project	# Issues	# Core Developers (Eigenvector)	# Core Developers (Hierarchy)	Observation period
Angular	38.679	4.470	4.470	2014-01-17 - 2020-10-06
Atom	21.194	4.133	4.112	2012-01-21 - 2020-12-24
Bootstrap	31.858	4.723	4.722	2011-08-19 - 2021-01-07
Deno	8.764	591	591	2018-05-29 - 2020-12-22
Moby	41.747	5559	5543	2013-10-31 - 2020-12-22
OpenSSL	11.161	631	631	2013-09-05 - 2020-02-27
React	20.264	3047	3045	2013-05-29 - 2020-12-23
Reveal.js	2.818	548	540	2011-06-07 - 2020-12-17
Tensorflow	45.704	7.004	7.004	2015-11-09 - 2020-12-26
VSCode	111.284	13.451	13.451	2015-10-13 - 2020-12-27

Angular[1] is a TypeScript-based open-source framework for building dynamic web applications. Atom[2] is a modern, hackable, and highly customizable text editor. Bootstrap[3] is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. Deno[4] is a runtime for JavaScript, TypeScript, and WebAssembly built on the V8 Javascript engine, the Google runtime engine for JavaScript and the Rust programming language. Moby[5] is an open-source project by Docker that provides a framework for assembling specialized container systems. OpenSSL[6] is a toolkit for general-purpose cryptography and secure communication. React[7] is an open-source JavaScript library for building user interfaces or UI components.
Reveal.js[8] is an open-source library for creating presentations built on open web technologies, including HTML, CSS, and JavaScript. TensorFlow[9] is an open-source machine learning framework. Visual Studio Code (VSCode)[10] is a lightweight, cross-platform source code editor.

1 https://angular.io/
2 https://atom.io/
3 https://getbootstrap.com/
4 https://deno.com/
5 https://mobyproject.org/
6 https://www.openssl.org/
7 https://react.dev/
8 https://revealjs.com/
9 https://www.tensorflow.org/
10 https://code.visualstudio.com/

3.3 Network Construction

In this section, we present how we construct developer networks for both cochange and issue data.
For both developer networks (Section 2.3.2), we use the same configuration to collect our core developers to ensure comparability of the results. The vertices are connected by undirected edges, and the network is simplified which helps us to compute our analysis faster without changing the results. We remove all authors who do not contribute to our specific network, so that only the developers who committed changes are present in the cochange network, and only the developers who participated in issues are present in the issue network and we do not have an overload of authors. This would result in a false classification of core and peripheral developers, so we would get developers with a low eigenvector or hierarchy centrality classified as core developers because there are so many developers with a value of zero.

The cochange developer network (Section 2.3.2.1) is built by using all the developers who have contributed a commit to the OSS project. Developers are connected by undirected edges with each other if they have contributed changes to the same file. And as described the network is simplified for faster computation on these networks.

The issue developer network (Section 2.3.2.2) is built by using all the developers/participants who have interacted with an issue. This includes not only comments but also any interaction such as opening issues, adding a tag, or closing issues. Developers are connected by undirected edges with each other if they have contributed in some way to the same issue. And as described the network is simplified for faster computation on these networks.

3.4 Analysis

Our first step is to classify the core and peripheral developers in the OSS project. There are different approaches to classify core and peripheral developers in an OSS project, we decide to do as Bock et al. [4] have done, as it proves to capture structural differences in developer relationships and has similar performance in detecting core developers and precision as the other classification metrics. We use the library coronet[11] for data processing, network construction, and centrality computation and use the eigenvector centrality value as our indication of a core developer. The developer's eigenvector centrality value shows how connected they are to other developers. The eigenvector centrality value of a developer is influenced by the weight of importance of the developers they interact with. This means that a developer has a high eigenvector centrality value by being connected to many developers, or by being connected to developers with a high eigenvector centrality value themself (Section 2.3.3). In addition, we use the hierarchy metrics to get another classification of core developers. The network hierarchy represents the community structure of a developer

11 https://se-sic.github.io/coronet/

network. The hierarchy centrality value of a developer is influenced by the amount of developers they are connected with and how they are connected to each other. This means that a developer has a high hierarchy centrality by being connected to many developers who are loosely connected to each other. A lower hierarchy centrality indicates a developer being connected to fewer developers with a tighter connection to each other (Section 2.3.4).

We then take our results of both classifications of core developers and analyze them separately. We take the cochange and issue core developer lists of one classification and join these two lists. The intersect of the lists are our core developers in both areas, the excepts of the two lists are our core developers of their respective areas. In addition, we intersect the lists of both classifications to get a list of core developers in both areas of both classifications.

After that, we analyze the developers' activities using different approaches depending on which group we place them in. First, we analyze the developer lists of both classifications separately. For the core developers of cochange data, we analyze their commits to understand what kind of contribution they make, such as bug fixing, refactoring, or feature implementation, how important their changes are, and how often and in what time periods they contribute. For the core developers of issue data, we analyze their participation in issues, such as commenting, opening or closing issues, reviewing or approving PR. This helps us to understand their role and importance in issues. For the core developers in both areas, we do both of these analysis steps to understand their role in both areas whether they are as important in one area as they are in the other, or whether they dominate one area more than the other.
For all the developer lists of both classification metrics, we do the same analysis in each area.
Furthermore, we take a closer look at core developers, who are classified as core in both issue and cochange data. This specific focus on this group aims to uncover potential subgroups or differences within the developers. By examining their activities, we gain a more nuanced understanding of their roles and contributions in the context of both code-related and issue-related aspects of OSS projects.
To achieve this, we use two key indices, one based on issue-related interaction and the other based on cochange-related contribution. These indices represent diverse aspects of a developer's engagement within the project. We aim to distinguish potential subgroups within the core developer group based on their distinct patterns of activity by combining information from both indices. These subgroups can form as a result of developers having different levels of involvement in code-related tasks, problem solving, or a balanced involvement in both areas. This dual-index approach allows for a comprehensive examination of the activities of core developers, providing a more comprehensive understanding of their multifaceted contributions to OSS projects.

This approach aligns with the recognition that the traditional model of classifying developers into core and peripheral categories is too abstract. Our goal is to delve into these diverse developer roles in detail, providing a more comprehensive and insightful view of the ecosystem.

3.5 Operationalization

In this chapter, we outline the operationalization for analyzing collaboration and interaction in the context of OSS development. The collaborative nature of development projects and the interactions among developers play a critical role in the success and evolution of these projects. To capture these dynamics, we use two indices: the collaboration index and the interaction index.

3.5.1 Collaboration Index

In the complicated environment of collaborative software development, understanding the dynamics of individual contributions is essential. It's not just about the quantity of code added or modified; it's about the content of each change, its impact, and how it aligns with the project's goals. The contribution index is a metric designed to capture the multifaceted nature of contributions in a comprehensible way.
A simple line count or commit frequency may not distinguish between routine tasks and more comprehensive contributions. The contribution index overcomes this limitation by evaluating the content of commit messages, recognizing specific keywords that identify significant actions, and assigning points accordingly.

3.5.1.1 *Evaluate Commit Messages*

First, we evaluate the commit messages. Each message is examined for keywords that indicate significant actions, such as adding a new feature or fixing a bug. Points are assigned based on the relative importance of these actions.
The full list of keywords used for the contribution index is shown in Section A.1.1.

3.5.1.2 *Contribution Index Calculation*

The contribution index is then calculated by summing up the points awarded to each developer, creating a comprehensive index that reflects their overall contribution. Additionally, we add an indicator to identify whether a developer has committed significant contributions. Significant contributions hereby are contributions with a greater impact on the project. We consider a contribution to be significant if its keyword gets weighted higher than 3.
The `calcContributionIndex` function takes a project as input, reads its commits and evaluates them, assigns points based on the commit, identifies significant contributions, and finally calculates the contribution index for each developer.

3.5.2 Interaction Index

The interaction index is a key index that measures the depth and breadth of a developer's engagement beyond just code contributions. Developers engage in discussions, handle issues, and facilitate the progress of projects.
While code contributions are an essential aspect of development, the social and communicative dimensions are equally important. The interaction index attempts to quantify a developer's participation in discussions, issues, and overall collaboration. This provides a more comprehensive view of their impact on the project.

3.5.2.1 *Interaction Index Calculation*

The interaction index is calculated based on the different types of interactions a developer has within a given project. Each type of interaction is assigned a weight that reflects its perceived impact on collaboration. The formula for calculating a developer's interaction index is as follows:

$$InteractionIndex = \sum_i Weight_i \times Count_i$$

Where:

- $Weight_i$ is the assigned weight for the interaction type i.
- $Count_i$ is the count of interactions of type i for the developer.

The full list of keywords used for the interaction index is shown in Section A.1.2.

The `calcInteractionIndex` function takes a project as input, reads its interactions, determines interaction types, assigns points based on weights, and finally calculates the interaction index for each developer.
This methodology provides a nuanced understanding of a developer's collaborative contributions, emphasizing the various ways in which developers engage with a project beyond just code commits.

3.6 Time-Based Analysis

In the dynamic landscape of OSS development, understanding how developers contribute over different time periods is crucial. In this chapter, we outline the operationalization for analyzing collaboration and interaction over time in the context of OSS development. To capture these dynamics, we use the two key indices, the collaboration index and the interaction index, but in a time-based variant.

3.6.1 Time-Based Contribution Index

In this section, we introduce the time-based contribution index, a metric designed to capture the nuances of developer contributions over different time periods.

3.6.1.1 *Time-Based Contribution Index Calculation*

The time-based contribution index is calculated by evaluating developer contributions within specific time periods. The entire project timeline is divided into distinct time periods of 9 months, and the contribution index is calculated separately for each time period. In addition, we add an indicator to determine whether a developer has made significant contributions. Significant contributions are contributions that have a greater impact on the project. We consider a contribution to be significant if its keyword gets weighted higher than 3.
The time-based contribution index provides insight into how developers' contributions shift over time, providing a nuanced understanding of their contribution patterns during different project phases. This approach makes it possible to identify trends over time and assess the importance of developers over different time periods.

3.6.2 Time-Based Interaction Index

In this section, we introduce the time-based interaction index, a metric designed to capture the nuances of developer interactions over different time periods.

3.6.2.1 *Interaction Index Calculation Over Time*

The time-based interaction index is calculated by considering different types of interactions a developer has during different time periods. Each type of interaction is assigned a weight that reflects its relative impact on collaboration during that time period. We divide the project into 9 month periods. The formula for calculating a developer's time-based interaction index is as follows:

$$Time - BasedInteractionIndex = \sum_i Weight_i \times Count_i$$

Where:

- $Weight_i$ is the assigned weight for the interaction type i.
- $Count_i$ is the count of interactions of type i for the developer in the time period.

The `calcInteractionIndex_timebased` function takes a project as input, divides its interactions into different time periods, assigns weights based on interaction types, and calculates the time-based interaction index for each developer within each time period.

The time-based interaction index addresses the limitation of a static view and the lack of insight into how a developer's collaboration patterns change over time by introducing a time-based dimension, allowing for a nuanced analysis of developer engagement at different stages of a project. This approach provides a dynamic understanding of how developers contribute to collaborative efforts over time, providing valuable insights for project management and fostering effective team dynamics.

4

Evaluation

In this chapter, we present the results of our network analyses of the ten different projects described in Section 3.2. Furthermore, we explore the comparison between the cochange and issue core developer groups we identify in the hierarchy and the eigenvector metrics. We review these results and discuss their contribution for answering our research question.

4.1 Results

In this section, we present the results of our network analyses an the characteristics of each core developer group of the ten different projects described in Section 3.2.

4.1.1 Analysis of Examined Projects

In this section, we detail the process of classifying core developer groups based on the cochange and issue networks using eigenvector and hierarchy centrality. Interestingly, our analysis reveals distinctive patterns in the composition of these core developer groups.
We observe that the classification using the hierarchy metric yields fewer core developers who belong to both the cochange and issue core groups, as shown in Table 4.1 and Table 4.2. This indicates a lower overlap in core developers when using the hierarchy metric compared to the eigenvector metric. Intriguingly, the cochange core developer group is approximately 1-15% the size of the issue core developer group in all examined projects.

Table 4.1: Number of recognized issue core developers, Number of recognized cochange core developers, Number of recognized core developers in both areas (issue and cochange) for the classification via eigenvector centrality

Project	# Core Developers (Issue)	# Core Developers (Cochange)	# Core Developers (Both)
Angular	4470	136	120
Atom	4133	60	48
Bootstrap	4723	44	36
Deno	591	70	66
Moby	5558	231	191
OpenSSL	631	84	63
React	3046	159	122
Reveal.js	540	28	19
Tensorflow	7005	304	233
VSCode	13451	201	157

Table 4.2: Number of recognized issue core developers, Number of recognized cochange core developers, Number of recognized core developers in both areas (issue and cochange) for the classification via hierarchy centrality

Project	# Core Developers (Issue)	# Core Developers (Cochange)	# Core Developers (Both)
Angular	4470	136	107
Atom	4112	60	48
Bootstrap	4722	44	35
Deno	591	70	53
Moby	5543	232	231
OpenSSL	631	84	61
React	3045	160	114
Reveal.js	548	28	11
Tensorflow	7004	304	230
VSCode	13451	201	141

This size difference between the issue core developer group and the cochange core developer group is visually represented in Figure 4.1. In this figure, the green area represents those core developers classified as core by both centrality metrics.

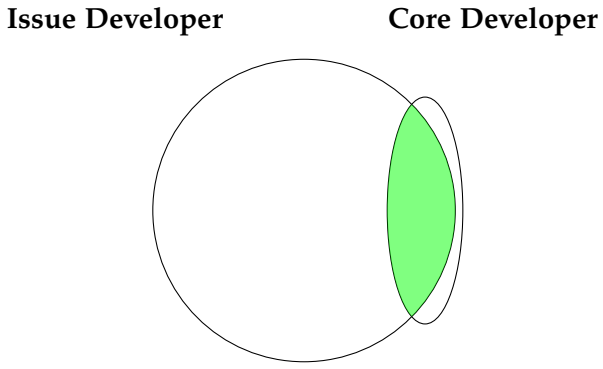

Figure 4.1: Overlap of Cochange and Issue Core Developer Groups

In Figure 4.2, the overlap of the cochange core developers between the two classification metrics is shown, the red area represents the core developers classified as core by both metrics. This figure shows how similar the cochange core developers were classified even by different metrics, as the overlap is enormous and they are nearly identical.

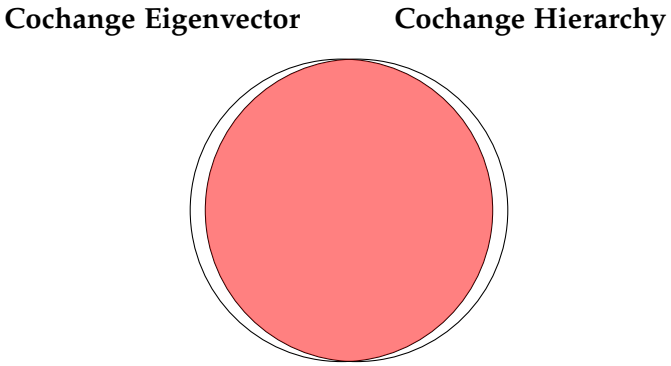

Figure 4.2: Overlap of Cochange Core Developer Groups Classified by Eigenvector and Hierarchy Metrics

The overlap of the issue core developer between the two classification metrics is shown in Figure 4.3. The blue area represents the core developers classified as core by both metrics. This figure shows how differently the issue core developers were classified by the different metrics.

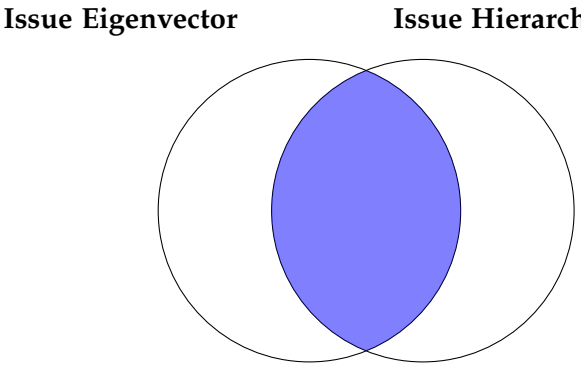

Figure 4.3: Overlap of Issue Core Developer Groups Classified by Eigenvector and Hierarchy Metrics

These visual representations emphasize the difference between the two core groups in terms of their classification based on the hierarchy and eigenvector metrics. These findings highlight the complex nature of core developer groups within OSS projects and provide a foundation for a more detailed exploration of their roles and responsibilities which we examine in Section 4.1.2.

Furthermore, we identify the most relevant issue core developers, typically constituting the top 20%, using both centrality metrics. These issue core developers overlap significantly, consistently appearing in both the core developer groups classified by eigenvector and hierarchy centralities.

Remarkably, in two of the examined projects (OpenSSL and Reveal.js), the core developer group classified by the hierarchy metric was entirely congruent with the core developer group obtained through the eigenvector centrality classification. Additionally, in these cases, the cochange core developer groups were identical, differing only in the issue core developer composition.

We identify that not all cochange core developers were equivalent, with approximately a

10% differentiation. Interestingly, these differences mainly occur within the lower 20% of the core developer group.
These results emphasize the influence of the centralities used in core developer identification and highlight the consistency in cochange core developers across different metrics. The divergence mainly lies within the issue core developer group, suggesting nuanced developer interactions and contribution patterns in OSS projects.

The previously present results do not apply to all the projects in our sample set. The results for Deno have some differences from our general results, which we present next.
Examining the issue core developers in Deno, we found a remarkable contrast to the previous projects. While in our findings the top 20% of issue core developers, as determined by the eigenvector metric, were also classified as issue core developers by the hierarchy metric, there was a remarkable difference in the Deno project. In this case, not even the top 1% of issue core developers, identified by the hierarchy metric, appear in the issue core developer group when applying the eigenvector metric.
These unique results indicate the significance of understanding how different projects can influence core developer patterns. This further highlights the complexity and variability in core developer identification and interaction within OSS projects.

4.1.2 Characteristics and Functions of each Core Developer Group

An integral part of our analysis is understanding the characteristics and functions of the two core developer groups, namely the cochange core and the issue core. Within the context of OSS projects, it's evident that much revolves around core developers, especially those who are core in both areas. This observation suggests that high involvement in one area often corresponds with significant involvement in the other, to the extent that the developer can be classified as core in both areas.
However, there are a few individuals who primarily focus their contributions on only one of these areas. While they may not be classified as core in both areas, their active participation in one area also influences their role in the other, and at least, ensures that they are recognized as core developers there.
In examining the responsibilities and functions of core developers, we found that those who are core in both areas often perform multiple tasks. These tasks typically include answering questions, so core developers who are core in both cochange and issue data actively engage in answering queries and providing solutions to issues raised by the community. Another task is implementing tests, so core developers who are core in both areas contribute to the quality assurance of the software. Furthermore, there is implementing comprehensive features and fixes, which are crucial to its advancement. Core developers who are core in both areas also focus on documenting the project. This documentation is essential for ensuring that others can understand, use, and contribute to the software.
In contrast, core developers who are only core in the cochange area tend to focus on tasks such as feature implementation, so they are primarily involved in implementing new features and functionalities. Also, they do refactoring tasks, bug fixing, ensuring the software's stability, and typo corrections in the code and documentation.

On the other hand, core developers who are solely core in the issue area predominantly participate in issue management so they are actively involved in managing issues raised by the community, which includes assessing, prioritizing, and ensuring these issues are addressed. They also actively play a vital role in reviewing and approving PR, ensuring code quality and compliance with project standards.
It's also worth noting that issue core developers are more likely to make contributions to the codebase related to issues and formatting, while cochange core developers focus primarily on larger code-related tasks.
These different roles and functions within the core developer groups illustrate the multi-faceted nature of OSS project collaboration.

We perform an in-depth analysis of three projects, to provide an overview and show the roles of core developers in OSS projects. This analysis introduces the various developer roles and responsibilities within these projects. We also looked at the other projects, but found similar results and patterns, so we have not included them here for the purpose of clarity.

Deno Project

In our analysis of the Deno project, we find different activity patterns for developers in all three core groups. We now present a small sample of these core developers in the different groups to represent their differences even though they are classified into the same group.

Table 4.3: Developer Tasks Deno Project

Developer	Tasks/Competencies
Core Developer 1	Reviewing PR, opening new PR, creating and resolving issues, implementing features, fixing bugs, refactoring code, improving performance, running tests, actively participating in discussions, and approving PR and new releases.
Core Developer 2	Active community member of the project, participating in issues and PR and reviewing PR.
Core Developer 3	Active contributor to the project, implementing new features, fixing bugs, and increasing performance.
Core Developer 4	Active in issues, implementing new features, and fixing bugs.
Core Developer 5	Actively fixes bugs.
Cochange Developer 1	Primarily focuses on implementing new features.
Cochange Developer 2	Primarily focuses on implementing new functionality and features.
Cochange Developer 3	Primarily focuses on fixing typos in the codebase.
Issue Developer 1	Reviewing PR, opening new PR, creating and managing issues to address problems and enhancements in the project, maintaining code formatting to ensure it meets the project's standards.
Issue Developer 2	Actively participates in the issue-tracking process by opening new issues, contributing to discussions, creating PR to propose solutions, reviewing multiple PR to ensure code quality. Responsible for implementing minor fixes and initiating discussions to drive collaboration within the project.
Issue Developer 3	Actively reviews PR, opens new PR, addresses issues, opens new issues, and implements necessary fixes to enhance the project's quality.

OpenSSL Project

In our analysis of the OpenSSL project, we find different activity patterns for developers in all three core groups. We now present a small sample of these core developers in the different groups to represent their differences even though they are classified into the same group.

Table 4.4: Developer Tasks OpenSSL Project

Developer	Tasks/Competencies
Core Developer 1	Contributing significantly to both code development and issue management. Reviewing PR, opening new PR, approving PR, creating and resolving issues, implementing features, fixing bugs, refactoring code, improving performance, engaging in discussions, and providing answers.
Core Developer 2	Active community member of the project. Responsibilities include participating in issues and PR.
Core Developer 3	Active contributor to the project. Actively implements new features and fixes issues.
Core Developer 4	Actively fixes bugs.
Core Developer 5	Actively participates in issues and implements features.
Cochange Developer 1	Primarily focuses on fixing typos in the codebase.
Cochange Developer 2	Primarily focuses on implementing new functionality and features, fixing issues, and maintaining the codebase's integrity.
Cochange Developer 3	Primarily focuses on implementing new functionality.
Issue Developer 1	Actively reviews PR and implements necessary functionality to enhance the project's quality.
Issue Developer 2	Actively reviews PR, addresses issues, and opens new issues.
Issue Developer 3	Primarily focuses on reviewing PR.

Reveal.js Project

In our analysis of the Reveal.js project, we find different activity patterns for developers in all three core groups. We now present a small sample of these core developers in the different groups to represent their differences even though they are classified in the same group.

Table 4.5: Developer Tasks Reveal.js Project

Developer	Tasks/Competencies
Core Developer 1	Contributing significantly to both code development and issue management. Reviewing PR, opening new PR, approving PR and new releases, creating and resolving issues, implementing features, fixing bugs, refactoring code, improving performance, running tests, participating in discussions, providing answers, and serving as the code owner.
Core Developer 2	Participates in issues and implements new features.
Core Developer 3	Implements new features.
Core Developer 4	Participates in issues.
Core Developer 5	Actively fixes bugs and refactors the codebase.
Cochange Developer 1	Primarily focuses on fixing bugs.
Cochange Developer 2	Contributes to the Reveal.js project with a unique responsibility. Focuses on updating copyright information and ensuring that the project adheres to legal and licensing requirements.
Cochange Developer 3	Actively works on adapting coding styles, configuring settings to fine-tune project behavior, and identifying and removing unnecessary controls.
Issue Developer 1	Actively addresses issues and tests project functions.
Issue Developer 2	Actively participates in issues and focuses on enhancing features.
Issue Developer 3	Actively participates in issues.

This overview shows the diverse roles and responsibilities of core developers within the OSS development community, and how core developers in these projects contribute to the success and advancement of their respective projects, further highlighting the importance of their work.

Notably, we discover a unique finding for the Reveal.js Project:
Reveal.js: The active core centers primarily around Core developer 1, whose contributions span the entire project and have a very broad role. Core developer 1 holds the distinct position of a code owner within the project, showing his influential status and far-reaching contributions.

4.2 Differentiate Developers of Core Group

In our analysis, we observe an interesting differentiation within the core developer group, especially those classified as core in both cochange and issue areas. As previously demonstrated, there is an overlapping group of developers who are core in both areas, indicating their significant involvement in both aspects of the project. However, further examination of this overlapping group reveals the existence of smaller, but highly influential subsets within this shared core group, using specific criteria for nuanced classification.

4.2.1 Group Classification Criteria

To capture the nuances of developers' contributions, we employ two key criteria: `median ratio` and `individual ratio`.

4.2.1.1 *Median Ratio Calculation*

The `median ratio` serves as a comparative metric, allowing us to identify developers with versatile contributions. By comparing the ratio of each developer's collaboration and interaction indices to the median values of their respective groups, we standardize the comparison. This helps identifying developers whose contributions span multiple aspects of the project, demonstrating well-rounded participation.

4.2.1.2 *Individual Ratio Calculation*

The `individual ratio` helps assess the dominance of a developer's activity in specific areas, facilitating their classification into distinct subgroups. Calculating the ratio of a developer's collaboration to interaction contributions and vice versa provides insight into their specialization. This criterion helps us identify developers who specialize in particular aspects of the project.
These criteria play a crucial role in the classification of developers, providing a nuanced understanding of their collaboration and interaction patterns. The `median ratio` allows us to identify versatile contributors, while the `individual ratio` helps to identify the dominant area of activity for developers, allowing their classification into specific subgroups. This approach enriches our analysis and provides deeper insights into the dynamics of core development within OSS projects.

4.2.2 Developer Subgroup Identification

Each of these subsets is characterized by unique patterns of contribution and interaction. We delve into the detailed characteristics of these subsets, shedding light on their distinct qualities and the methods used to accurately identify them. We will refer to these smaller, highly influential subsets as:

4.2.2.1 *Versatile Contributors*

As true all-rounders, versatile contributors are highly multifunctional, covering a wide range of tasks in all areas of the project. They may have particular strengths in specific task areas, highlighting their flexibility and adaptability to different project situations. They are usually the project's top developers with the highest level of competence. Versatile contributors show a balanced involvement in both collaborative efforts and code contributions. They

are identified by having collaboration index and interaction index values above a certain threshold. Developers are considered versatile when both their `median ratio` of the collaboration index and interaction index are greater than 5. This criterion ensures a well-rounded contribution to both collaborative discussions and code development.

4.2.2.2 *Comprehensive Developers*

Comprehensive developers not only handle bug fixes and feature implementation but are also actively involved in issues. While they may overlap with the feature-focused developers, they distinguish themselves by actively participating in code-related improvements and covering a broader range of tasks. Comprehensive developers contribute in a balanced way across code and collaboration, without specializing in a particular area. Developers are classified as comprehensive if they do not fall into the bug fix specialist, feature-focused, social connector, or versatile categories. They are also identified based on their `individual ratio` of the interaction index and collaboration index which lie between 0.5 and 1.5. This ensures a broad and balanced contribution pattern.

4.2.2.3 *Social Connectors*

Social connectors are characterized by active participation in social interactions within the project. They participate in discussions related to PR and issues and significantly contribute to building a sense of community within the project. Social connectors are developers who actively participate in collaborative interactions, but may not contribute as much to the codebase. They are identified based on their `individual ratio` of the interaction index and collaboration index. Developers are classified as social connectors if their `individual ratio` of the interaction index is greater than 2.

4.2.2.4 *Feature-Focused Developers*

Feature-focused developers primarily specialize in feature development. In addition to feature development, they also focus on fixing issues (Fixes). Some of them focus on a specific task of their choice, such as improving tests or improving project performance. Developers in this group contribute significantly to the codebase, but may not be as active in collaborative discussions. They are identified based on their `individual ratio` of the interaction index and collaboration index. Developers are classified as feature-focused if their `individual ratio` of the collaboration ratio is greater than 2 and their contributions are at least 10% significant contributions.

4.2.2.5 *Bug Fix Specialists*

Bug fix specialists focus primarily on identifying and fixing issues within the project. They are also actively involved in the refactoring process and have a keen eye for correcting

typos. Bug fix specialists prioritize fixing issues and bugs, maintaining a balance between collaborative efforts and code contributions. They are identified by their `individual ratio` of the interaction index and collaboration index. Developers are classified as bug fix specialists if their `individual ratio` of the collaboration index is greater than 1.5, and their collaboration index is less than twice the median collaboration index.
This revelation underscores the complicated nature of the core development dynamic within OSS projects, and highlights the essential role played by these key contributors. The presence of such influential figures within the core group enriches our understanding of OSS development.

4.2.3 Group Classification - Time-Based Approach

Classifying developers into different groups is a critical step in understanding their roles and contributions to a project. In the time-based approach, we use two key metrics: the collaboration index and the interaction index. These indices capture a developer's involvement in code contributions and collaborative interactions, respectively. Therefore, we use the indices and ratios shown above in a time-based variant.

4.2.3.1 *Calculation of Indices*

The collaboration index is calculated by using the `calcContributionIndex_timebased` function, which analyzes code contributions over different time periods. Similarly, the interaction index is calculated using the `calcInteractionIndex_timebased` function, which evaluates the variety and intensity of a developer's collaborative interactions over time.

4.2.3.2 *Calculation of the Ratios*

In addition to the modified `calcContributionIndex_timebased` and `calcInteractionIndex_timebased` functions, we also modify the `median ratio` and `individual ratio` with the time based functions of the two indices. This way, we can use these time-based ratios later for group classification.

4.2.3.3 *Group Classification*

To classify developers into different groups, we use two sets of different metrics: the `median ratio timebased` and the `individual ratio timebased`. These ratios allow comparisons between developers and help in the identification of versatile contributors and those with specific focuses.
In the code above, the `group_classification_timebased` function iterates over different time periods, calculates `median ratio timebased` and `individual ratio timebased`, and identifies developers who belong to different groups such as versatile contributor, social connectors, feature-focused, bug fix specialists, and comprehensive developers.

This classification provides insight into how developer roles evolve over time, providing a dynamic perspective on their contributions and areas of focus on a project.

4.2.4 Results

In this section, we present our results for the developer distribution of the identified subgroups.

Table 4.6: Developer Distribution Across Projects

Project	Versatile	Comprehensive	Social	Feature	Bug Fix
Angular	31	27	44	14	4
Atom	6	5	36	1	0
Bootstrap	10	4	22	0	0
Deno	10	30	11	12	3
Moby	31	60	82	16	2
OpenSSL	17	16	26	3	1
React	23	33	52	11	1
Reveal.js	2	9	6	1	1
Tensorflow	11	73	40	100	6
VSCode	31	29	86	9	2

In the exploration of developer dynamics within OSS projects, our analysis sheds light on the nuanced roles and contributions of core developers. Delving into the specific details of eight different projects - Deno, OpenSSL, Reveal.js, Angular, Atom, Bootstrap, TensorFlow, VSCode, Moby, and React - we find distinct patterns in the distribution and behavior of versatile contributors, comprehensive developers, social connectors, feature-focused developers, and bug-fix specialists. We find that versatile developers have a moderate representation in each project. Comprehensive developers also have a moderate representation with peaks in contribution-heavy projects such as Moby and Tensorflow. Social connectors are likely to be relatively numerous in each project. The number of feature-focused developers could vary, as projects require different levels of feature development. As observed, bug fix specialists tend to have a smaller representation. This aligns with the expectation that not all core developers exclusively focus on fixing issues, and there's a strategic allocation of resources for bug resolution.

4.2.5 Results (Time-Based Approach)

In this section, we present our results from the time-based approach. We analyze the distribution of developers over different time periods. We also examine the transitions between the different subgroups of developers who join the project and those who return to the project after at least one period of inactivity. Additionally, we investigate those developers who leave the project after one period and separate them into those who leave for at least 1 period and come back, and those who leave the project completely.

Angular Project

Angular shows a dynamic distribution of core developers over time, with noticable peaks in both comprehensive developers and social connectors. This variability suggests a project that values a diverse range of contributions, with periods of increased collaboration and community engagement. Particularly noteworthy is the crucial role played by feature-focused developers, especially during phases of intense feature development. Examining the patterns of newcomers, it becomes clear that they distribute across all groups except versatile contributors. Interestingly, returners consistently gravitate to social connectors, suggesting a tendency for previously engaged developers to reconnect with a wider range of tasks or community-oriented tasks. Furthermore, the observed transitions of feature-focused developers into comprehensive developers and comprehensive developers into social connectors highlight a pattern where contributors move from specialized feature development to broader code-related improvements and community engagement, contributing to the overall vibrancy and adaptability of the Angular project. We can observe that most core developers who leave the project after a period of time leave the core group completely, and a small number of comprehensive developers and social connectors return to the project in later time periods. Notably, core developers never leave the core group as versatile contributors.

Table 4.7: Developer Distribution Over Time - Angular Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	0	3	0	9	0	N.A.	N.A.	N.A.	N.A.
2	0	9	12	6	0	16	0	1	0
3	3	10	6	7	0	13	0	11	3
4	1	9	7	2	0	5	0	10	2
5	0	5	17	2	1	7	3	4	0
6	1	7	17	0	0	9	2	12	0
7	1	7	15	0	1	7	0	6	2
8	0	1	18	1	0	5	1	10	0
9	0	0	2	0	0	0	0	19	0

Going into \ Coming From	Versatile	Comprehensive	Social	Feature	Fixer	Newcomer	Returner
Versatile	2	0	2	2	0	0	0
Comprehensive	1	10	5	8	0	21	0
Social	3	20	39	7	0	27	7
Feature	0	2	1	3	0	12	0
Fixer	0	0	0	0	0	2	0
Leaver Complete	0	15	48	8	2	0	0
Leaver Temp	0	1	6	0	0	0	0

Figure 4.4: Developer Transitions Over Time - Angular Project

Atom Project

Atom shows a dynamic distribution pattern characterized by alternating periods of comprehensive developers and social connectors. This dynamic variation suggests a project environment that experiences shifts in focus between code-related improvements and community engagement. Examining the patterns of newcomers reveals a preference for joining social connectors or comprehensive developers, with a noteworthy tendency toward social connectors. Notably, transitions into social connectors are most common, suggesting a tendency for contributors, regardless of their initial role, to tend toward community-related responsibilities. We can observe that most core developers who leave the project after a period of time leave the core group completely, and a small number of social connectors return to the project in later time periods. Interestingly, core developers never leave the core group as versatile contributors or feature-focused developers.

Table 4.8: Developer Distribution Over Time - Atom Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	0	0	0	3	0	N.A.	N.A.	N.A.	N.A.
2	0	3	0	0	0	1	0	0	0
3	0	2	3	0	0	2	0	0	0
4	1	0	8	0	0	5	0	2	0
5	1	1	12	0	0	8	0	3	0
6	0	0	11	0	0	4	1	8	0
7	2	2	5	0	0	5	0	6	1
8	0	1	7	0	0	3	0	5	0
9	1	0	4	0	0	3	0	6	1
10	0	1	1	0	0	1	0	5	0
11	0	0	1	0	0	1	0	3	1
12	0	0	0	0	0	0	0	2	0

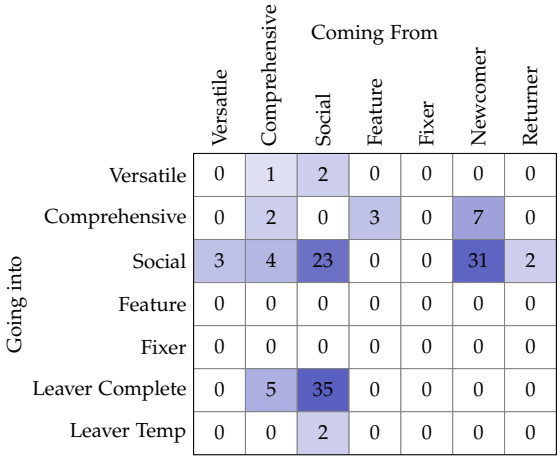

Figure 4.5: Developer Transitions Over Time - Atom Project

Bootstrap Project

Bootstrap demonstrates a consistent and stable presence of social connectors throughout its development timeline. When analyzing the influx of newcomers to the Bootstrap project, a clear trend can be seen, with the majority entering the social connectors. Returners show a similar trend, with a majority rejoining the social connectors. Notably, transitions between groups lead contributors mostly into the social connectors, highlighting the project's focus on maintaining a robust and engaged community. We can observe that most core developers who leave the project after a period of time leave the core group completely, and only a small number of comprehensive developers and social connectors return to the core group in later time periods.

Table 4.9: Developer Distribution Over Time - Bootstrap Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	0	1	1	0	0	N.A.	N.A.	N.A.	N.A.
2	0	0	3	0	0	1	0	0	0
3	0	0	2	0	0	0	0	1	0
4	0	1	4	0	1	4	0	0	0
5	0	0	6	0	0	2	0	3	0
6	0	0	4	0	0	0	0	2	0
7	0	0	2	0	0	0	0	1	1
8	1	0	3	0	0	3	0	1	1
9	0	0	4	0	0	1	1	3	0
10	0	0	3	0	0	0	0	2	0
11	0	0	2	0	0	0	0	1	0
12	0	0	1	0	0	0	1	0	1
13	1	0	2	0	0	2	0	2	0

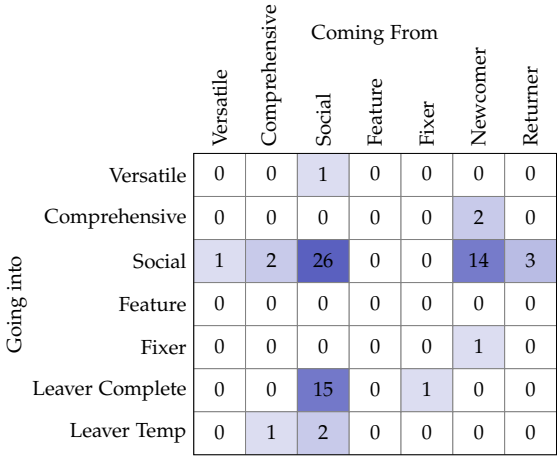

Figure 4.6: Developer Transitions Over Time - Bootstrap Project

Deno Project

We observe a notable increase in the number of versatile contributors and comprehensive developers over time, reflecting a project environment that values a diverse skill set and encourages broader contributions. This increase underscores the adaptability of developers, indicating a willingness to take on different roles across the project. At the same time, the social connector group shows stability, maintaining a consistent presence across all time periods, indicating the project's ongoing commitment to the community. An interesting aspect of the project's dynamics is the regular influx of newcomers in each period. These newcomers integrate seamlessly into various developer groups except versatile contributors. Versatile contributors tend to stay within their group, demonstrating a preference for diverse contributions. Comprehensive developers show a mix of staying in their current role and transitioning to versatile contributors or social connectors, demonstrating their versatility. Social developers have fewer transitions. Feature-focused developers tend to integrate into comprehensive developers, similarly to the focus on broader project contributions. In contrast, bug fix specialists transit in the social connector group. We can observe that all core developers who leave the core group after a period of time leave the core group completely, and do not return to the core group in later time periods. Notably, core developers never leave the core group as versatile contributors.

Table 4.10: Developer Distribution Over Time - Deno Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	1	11	2	0	0	N.A.	N.A.	N.A.	N.A.
2	1	6	3	3	0	6	0	7	0
3	4	21	4	5	2	27	0	4	0
4	4	6	9	2	1	8	0	22	0

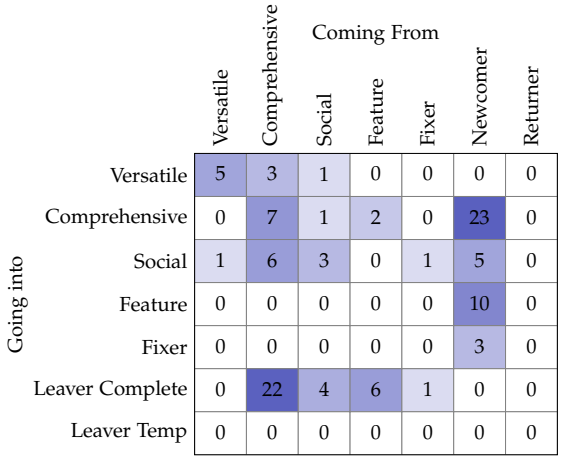

Figure 4.7: Developer Transitions Over Time - Deno Project

Moby Project

The Moby project shows a diverse and balanced distribution of developer types across its development stages. Notable peaks in comprehensive developers and social connectors signify a project that values both a broad skill set and community engagement. Feature-focused developers strategically join in during periods of significant feature development. The influx of newcomers into every developer group indicates an inclusive environment, with contributors finding their niche, particularly among comprehensive developers and social connectors. Returners often rejoin as social connectors, indicating a trend of particular active community re-engagement. Versatile contributors stay as versatile or transition into social connectors, underscoring the link between versatile skills and community engagement. Comprehensive developers exhibit a wide range of transitions, contributing to every group except bug fix specialists, showcasing their adaptability and diverse roles within the project. Social connectors primarily stay as social connectors. Feature-focused developers mostly transition into comprehensive developers or social connectors. We can observe that most core developers who leave the core group after a period of time leave the core group completely, especially feature-focused developers and bug fix specialists, and only a small number of versatile contributors, comprehensive developers, social connectors, and feature-focused developers return to the core group in later time periods.

Table 4.11: Developer Distribution Over Time - Moby Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	4	4	3	3	0	N.A.	N.A.	N.A.	N.A.
2	5	12	13	4	0	27	4	7	0
3	3	12	25	1	3	32	2	22	0
4	1	15	26	3	0	23	1	19	3
5	1	8	26	5	0	13	2	16	3
6	2	10	24	4	1	14	6	16	1
7	2	6	14	5	1	10	3	22	2
8	5	3	12	1	1	7	8	15	1
9	2	1	13	1	0	6	3	12	0
10	1	1	6	1	0	4	1	10	2
11	1	1	4	0	1	2	3	6	0

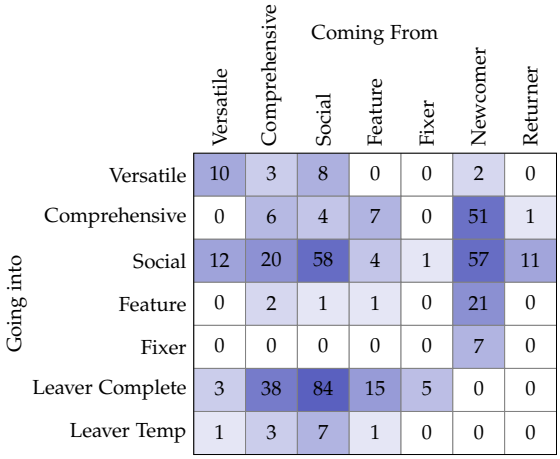

Figure 4.8: Developer Transitions Over Time - Moby Project

OpenSSL Project

The project has experienced a temporal increase in both comprehensive and feature-focused developers (e.g. during period 2-4), signaling a period of intensive development, code improvements, and new features. Throughout these phases, social connectors have consistently played a key role, maintaining a robust presence and actively encouraging community engagement. It is interesting that the number of versatile contributors increased over time, indicating a need for contributors with diverse skill sets capable of engaging in various project tasks. In addition, the project showed a consistent influx of newcomers in each time period, with these newcomers predominantly joining the comprehensive or social groups. This pattern underscores the project's openness to new contributors and the flexibility of these individuals to participate in different roles. As for returners, their reintegration into the project occurrs primarily in the social connector group. Notably, there were only a few transitions away from the social connectors, with a significant majority choosing to stay. Among comprehensive developers, most move to the social connectors, underscoring the collaborative and community-oriented nature of their contributions. Similarly, feature

developers often integrated with the comprehensive developers, matching the broader focus on project-wide improvements. We can observe that most core developers who leave the core group after a period of time leave the project core group, and a small number of comprehensive developers and social connectors return to the core group in later time periods. Notably, core developers never leave the core group as versatile contributors.

Table 4.12: Developer Distribution Over Time - OpenSSL Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	0	1	0	1	0	N.A.	N.A.	N.A.	N.A.
2	0	3	0	1	0	3	0	1	0
3	0	1	0	3	0	3	0	2	1
4	1	8	1	3	0	8	1	0	0
5	1	6	7	0	0	4	0	2	1
6	1	3	12	0	0	7	0	4	1
7	1	2	12	0	0	3	0	2	2
8	1	2	12	0	0	6	1	6	1
9	2	2	11	0	1	5	2	6	0
10	0	0	5	0	0	0	1	12	0

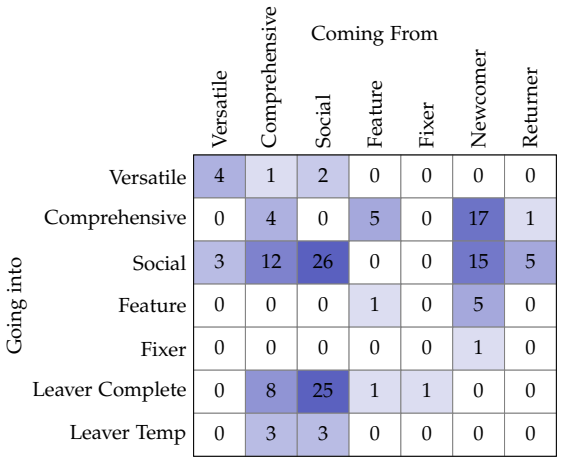

Figure 4.9: Developer Transitions Over Time - OpenSSL Project

React Project

The React project shows a diverse and balanced distribution of developer types across its development stages. The influx of newcomers into every developer group indicates an inclusive environment, with contributors finding their niche, particularly among comprehensive developers, social connectors, and feature-focused developers. Returners tend to rejoin as social connectors, indicating a trend of particularly active community re-engagement. Versatile contributors stay as versatile or transition into social connectors, underscoring the link between versatile skills and community engagement. Comprehensive developers exhibit a wide range of transitions, contributing to every group except bug fix specialists. Social connectors primarily stay as social connectors or transition into versatile contribu-

tors. Feature-focused developers mostly transition into comprehensive developers or social connectors. We can observe that most core developers who leave the core group after a period of time leave the core group completely, especially comprehensive developers and feature-focused developers. We also find only a small number of comprehensive developers, and social connectors return to the core group in later time periods.

Table 4.13: Developer Distribution Over Time - React Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	2	5	2	2	0	N.A.	N.A.	N.A.	N.A.
2	1	2	8	1	0	5	1	4	0
3	3	13	4	2	1	15	3	4	0
4	4	8	4	1	0	10	3	15	1
5	1	3	8	2	0	8	2	11	1
6	2	5	10	2	0	9	3	5	0
7	3	6	10	1	0	8	3	7	0
8	3	5	6	4	0	10	3	12	0
9	1	5	7	1	0	4	1	8	0
10	5	7	6	2	0	14	5	8	0
11	0	0	2	0	0	0	0	18	0

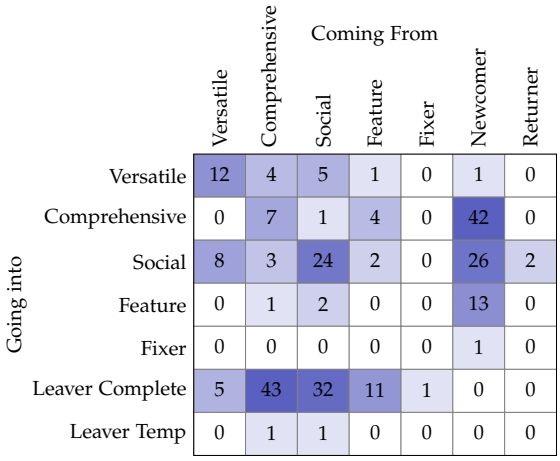

Figure 4.10: Developer Transitions Over Time - React Project

Reveal.js Project

The Reveal.js project showed notable stability in its developer distribution over time, with limited changes observed in the composition of core developer roles. Despite this constancy, there was a consistent presence of social connectors across time periods. The trend among newcomers is interesting, with the majority of newcomers joining either the comprehensive developers or the social connectors. In addition, the Reveal.js project showed a distinctive pattern of almost no transitions between core developer groups, indicating a certain level of role stability and specialization among contributors. We can observe that all core developers who leave the core group after a period of time leave the core group completely, and do not

return to the core group in later time periods. Notably, core developers never leave the core group as versatile contributors.

Table 4.14: Developer Distribution Over Time - Reveal.js Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	0	1	1	0	0	N.A.	N.A.	N.A.	N.A.
2	0	0	1	0	0	0	0	1	0
3	0	0	1	0	0	0	0	0	0
4	0	1	1	0	0	1	0	0	0
5	0	1	1	0	0	1	0	1	0
6	1	1	1	0	0	2	0	1	0
7	0	0	2	0	0	1	0	2	0
8	0	0	2	0	0	1	0	1	0
9	0	0	1	1	0	1	0	1	0
10	0	0	2	0	0	1	0	1	0

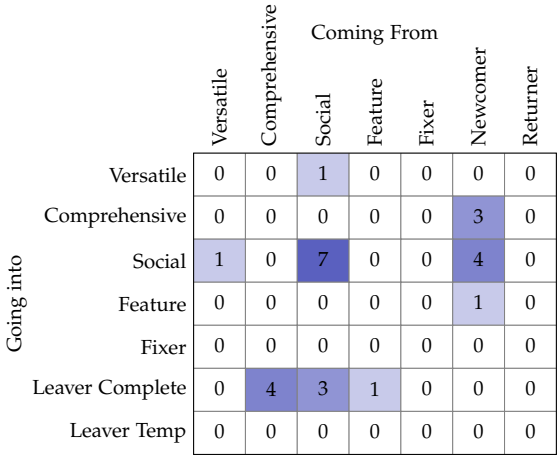

Figure 4.11: Developer Transitions Over Time - Reveal.js Project

Tensorflow Project

TensorFlow shows a dynamic distribution of core developers over time, with notable peaks in both comprehensive developers and social connectors. This variability suggests a project that values a diverse range of contributions, with periods of increased collaboration and community engagement. Particularly noteworthy is the crucial role played by feature-focused developers, especially during phases of intense feature development. The persistent presence of social connectors underscores TensorFlow's strong commitment to community engagement, with consistent efforts to encourage collaboration and interaction among contributors. The relatively low numbers of versatile contributors suggests a group of developers with diverse skills, comparable levels of engagement, and a balance of active and less active contributors. Analyzing the patterns of newcomers, a notable trend appears, with contributors entering each developer group, with the highest numbers entering as comprehensive and feature-focused developers. Returners, on the other hand, tend to find

their place as social connectors, indicating a tendency to revisit and actively engage with the community. Transition patterns show that comprehensive developers often move into social connectors, highlighting the connection of development and community engagement. Social connectors tend to stay within their group, demonstrating the project's ability to retain engaged community members, while feature-focused developers often move into comprehensive developers. We can observe that most core developers who leave the core group after a period of time leave the core group completely, and a small number of comprehensive developers, social connectors, and feature-focused developers return to the core group in later time periods. Notably, core developers never leave the core group as versatile contributors.

Table 4.15: Developer Distribution Over Time - Tensorflow Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	1	10	1	7	0	N.A.	N.A.	N.A.	N.A.
2	0	14	6	7	0	16	0	6	2
3	0	8	12	14	0	18	1	10	2
4	0	13	17	27	0	33	3	12	3
5	0	29	22	29	2	42	3	15	4
6	0	28	23	20	0	24	4	34	6
7	0	22	23	11	1	16	6	36	0

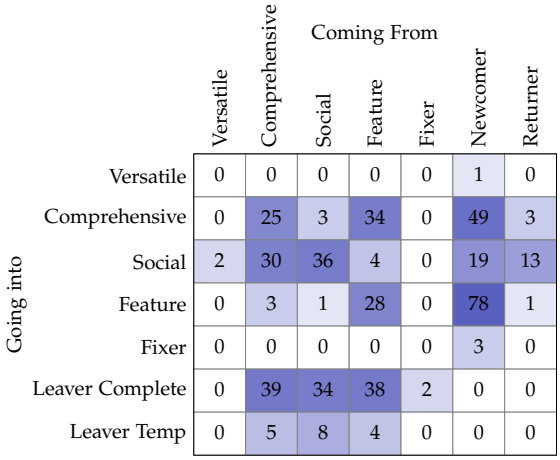

Figure 4.12: Developer Transitions Over Time - Tensorflow Project

VSCode Project

VSCode shows a dynamic distribution of core developers over time, with prominent peaks in both comprehensive developers and social connectors. This variability suggests a project that values a diverse range of contributions, with periods of increased collaboration and community engagement. Particularly noteworthy is the crucial role played by feature-focused developers, especially during phases of intense feature development. The entry of newcomers into every developer group except versatile contributors indicates a welcoming environment, with contributors finding their niche, especially in comprehensive developers

and social connectors. Returners, on the other hand, tend to rejoin as social connectors, indicating a trend of active community re-engagement. Versatile contributors often stay or move into social connectors, highlighting the connection between versatile skills and community engagement. Comprehensive developers display a wide range of transitions, contributing to every group except bug fix specialists, showcasing their adaptability and diverse roles within the project. Social connectors transition primarily into versatile contributors, reflecting the project's ability to channel social engagement into versatile skills and collaborative contributions. We can observe that most core developers who leave the core group after a period of time leave the core group completely, and only a small number of social connectors and comprehensive developers return to the core group in later time periods. Notably, core developers never leave the core group as versatile contributors.

Table 4.16: Developer Distribution Over Time - VSCode Project

Period	Versatile	Comprehensive	Social	Feature	Bug Fix	Newcomers	Returners	Leaver	Leaver Temp
1	0	11	5	2	0	N.A.	N.A.	N.A.	N.A.
2	4	9	12	1	2	16	0	5	1
3	6	7	20	1	0	14	1	9	0
4	15	9	20	0	1	22	0	10	2
5	9	7	20	3	0	17	0	21	1
6	9	4	24	2	0	15	3	19	0
7	5	5	23	0	0	10	0	17	0

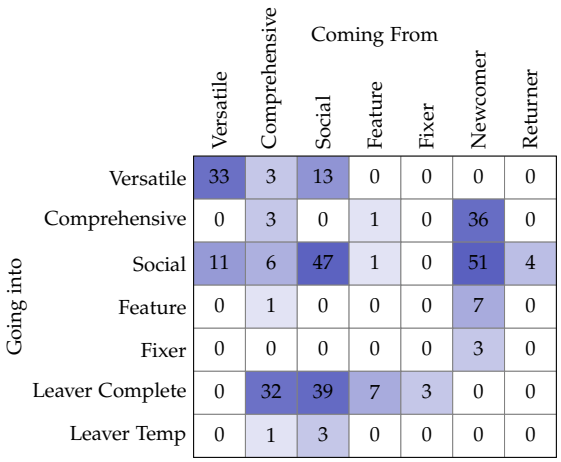

Figure 4.13: Developer Transitions Over Time - VSCode Project

4.3 Discussion

The in-depth analysis of core developer distribution across projects and over time has revealed interesting patterns and dynamics within OSS projects. In this discussion section, we delve into the nuanced interpretation of these findings and consider their implications.

4.3.1 Across Projects

OSS projects, as our analysis reveals, exhibit dynamic patterns of core developers shaped by project-specific needs that reflect the unique demands and priorities of each project. The specific needs and priorities of each project influence the distribution of core developers across subgroups. The different sizes of the core developer subgroups underscore the diverse nature of contributions within these projects. This diversity, which includes versatile contributors, comprehensive developers, social connectors, feature-focused developers, and bug-fix specialists, represents a collaborative landscape in which core developers with different strengths collectively contribute to the overall success of the project. The presence of social connectors in all projects underscores the importance of community interactions and represents a thriving environment for collaboration.
In summary, the distribution of core developers across categories reflects the multifaceted nature of OSS projects. The distribution of core developers is a dynamic balance between project-specific needs and the diversity of contributions. This complexity contributes to the dynamics and sustainability of OSS projects.

4.3.2 Over Time

The nuanced analysis of core developer distribution across subgroups reveals interesting insights into the multifaceted nature of OSS projects. The diversity of subgroup sizes indicates a diverse contribution landscape, where core developers representing different strengths and focuses collectively contribute to the overall success of the project. A larger number of social connectors underscores the central role of community interactions, suggesting that projects have a strong sense of community.
Our analysis of transitions between core developer subgroups over time reveals several patterns.
An interesting observation is the dynamic nature of core developer roles over time. Core developers show remarkable adaptability, transitioning over different time periods between subgroups. This dynamic pattern suggests a high degree of flexibility among core developers, allowing them to align their efforts with the evolving needs of the project.
Versatile contributors come primarily from comprehensive developers and social connectors. They mostly stay in their current roles or move to social connectors. Comprehensive developers show a balanced transition dynamic, moving proportionally from and to different groups. Social connectors appear to be a stable group, with the majority choosing to stay in this role. They attract core developers from the comprehensive developers, and those who leave the social connectors often transit into versatile contributors. Feature-focused

developers show a unique pattern with a continuous influx of newcomers, making it the most dynamic group with no clear pattern of departures across all projects. Surprisingly, in most projects, the majority either stay or evolve into comprehensive developers. In contrast, bug fix specialists consistently appear as newcomers and rarely stay beyond one period, resulting in a lack of recognizable transition patterns. This nuanced understanding of core developer transitions adds a temporal dimension to our analysis and sheds light on the dynamic nature of their roles within OSS projects.
An observation of the patterns of newcomers reveals exciting dynamics within developer groups. Bug fix specialists consistently appear, as already mentioned, as newcomers, with a consistent trend of short-term participation. In contrast, other core developer groups show diverse patterns with no clear linear trend. However, a notable trend is the relative strength of newcomers in social connectors and comprehensive developers, which varies according to project dynamics. Furthermore, investigating returners reveals that a significant amount of returning core developers move into the social connector role.
We also track core developers who leave the project for at least one time period and find some interesting patterns. We find that core developers are more likely to leave the project completely instead of leaving temporarily and returning later. In addition, we find that most versatile contributors do not leave the project, but rather stay in the project or move to another group before eventually leaving. We are also noticing that the core developers who do return to the project tend to leave the project from the comprehensive developers and social connectors groups.
Project-specific considerations emphasize the relation between developer distribution and the unique nature of each project. The observed distribution of core developers appears to be related to the unique nature of each project. In particular, the presence of feature-focused developers in TensorFlow shows their tendency to contribute heavily. In contrast, we have the consistent presence of social connectors in community-centric projects such as VSCode. The central role of social connectors in OSS projects cannot be overstated. The high number of social connectors in projects such as VSCode and Bootstrap underscore the importance of community interactions.

In summary, the time-based analysis of core developer distribution reveals dynamic patterns in each project. By examining how versatile contributors, comprehensive developers, social connectors, feature-focused developers, and bug fix specialists contribute over time, we gain insight into the evolving, dynamic, and multifaceted nature of OSS projects. The observed patterns offer valuable information about core developers and their different activity patterns for project maintainers and contributors aiming to understand and improve the sustainability of their projects and reflect the complex and evolving needs of each project. This research provides a solid foundation for future studies and offers practical insights in OSS environments.

4.4 Threats to Validity

In this section, we discuss the threats to internal and external validity of our research.

4.4.1 Internal Validity

The first threat to mention is the data we use to build our networks. Our developer networks are built using the commits and issues we extract from GitHub. This is a problem because some projects may not have issue data for the same observation period as the commit data, and may also use different channels of communication. As a result, some developers may be misclassified. We also rely on the correctness of our extracted data, and that all the authors we collect are unique and we do not have duplicates. In our analysis of the transitions between the subgroups, newcomers, returners, and the two types of leavers, we only consider those who are core developers in both areas. Thus, there may be developers who are still active in the project, but not contributing enough to be recognized and included in our analysis. This must be noted and taken into account to validate our results. Nonetheless, all of our projects have significant overlapping observation periods as well as representative data that we can use for our research.

4.4.2 External Validity

Our results are expected to apply to OSS projects of similar size. Nonetheless, there are OSS projects with a higher commit activity and issue participation such as Linux[1], which can have different results than our research provides. There are also OSS projects which are significantly smaller and have less commit activity and issue participation than ours. Since constructing developer networks requires a relatively high number of commits and issues, our results are not applicable to small-scale projects.

1 https://www.linux.org/

5

Related Work

As software projects continue to grow so does the understanding and importance of the involvement and roles of developers. In this chapter, we present related work with similar research questions about the characterization of developer groups, especially the core developers, in software projects. We are specifically interested in their patterns of activity and interaction with other developers. With this state of research, we can identify knowledge gaps and fill them with our research.

There are various approaches and studies on how to characterize developer roles and activities in software projects. The most relevant ones for us have investigated how to classify developers as core and peripheral, how their reputation influences code reviews, and how their activities can be measured and improved [1, 4, 5, 8, 12, 16, 17, 19, 20, 25].
For example, Oliva et al. [19] investigated the role of key developers in software projects using the Apache Ant[1] project as a case study. Developers get identified by their contributions and interactions to indicate their importance to the project. They have analyzed developer roles by their behaviors in software development and their importance in the contribution to project advancement.
Joblin et al. [12] worked on classifying developers into core and peripheral based on count and network measurements. They investigated different operationalizations of developer roles and how they affect developer networks. So in this paper, they analyze the developer roles, specifically core and peripheral, and their dynamics in software projects.
Also, Oliva et al. [20] investigated the identification and characterization of key developers using the Apache Ant project as a case study. Developers were analyzed based on their contributions and social interactions. The results suggest that a consistent team of core developers is essential.

We build developer networks to identify core developers and their connections to both other core developers and peripheral developers. We focus on capturing, analyzing, and modeling developer networks. We use social network analysis to understand the dynamics of collaboration and visualize the structure of these networks for better understanding and analysis [5, 7, 10, 11].
For example, Jermakovics et al. [10] present an approach to mine and visualize developer networks from version control systems. They compute similarities among developers in their file changes by creating developer networks and improving network modularity. Their focus is to visualize and analyze developer networks in order to improve our understanding of collaboration patterns.

1 https://ant.apache.org/

Bock et al. [5] introduce a method for analyzing temporal collaboration structures in communication and development within software projects. They provide insights into developer interaction patterns by examining the strength and stability of social subgroups through tensor decomposition. This allows for the study of the dynamics of developer communication and contribution, leading to a better understanding of these structures.

Yamashita et al. [28] show that most of the contributions are made by a small core developer team. However, it is indicated that most projects on GitHub do not follow the Pareto principle – roughly 80% of the code contributions are produced by 20% of the active developers. They indicate that there would be a harmful impact if one of the core developers leaves the project.

Joblin et al. [14] investigated a method to capture and visualize developer networks in software projects. The method uses similarities based on similar code changes to create a network of cooperating developers.

Robles et al. [22] investigated the evolution of the core developer team in a software project. They used data from version control systems to identify the most active developers in different time periods and analyzed their behavior over time to show how the core developer team evolves in their activities.

6 Concluding Remarks

6.1 Conclusion

In this thesis, we analyze the characteristics and activity patterns of core developers in OSS projects to understand their responsibilities and roles better. In particular, we look at the technical and social core developers as classified through commit and issue data, to find out if they are particularly important in this area or if they are less specialized in the technical and social area than the core developers of only one area. For this analysis, we build two different types of developer networks, one with the cochange data and the other one with the issue data. In these two networks, we classify developers by eigenvector and hierarchy centrality into core and peripheral to capture all core developers in each area. We divide these two classifications of core developers into three groups each to classify the core developers of both technical and social areas and those who are only core developers in one of them. Additionally, we take a closer look at core developers in both technical and social areas and find five subgroups with different characteristics and activity patterns, allowing us to differentiate core developers in a more nuanced way.

For both classifications, we find the core developers of both technical and social areas rank higher in terms in centrality to the network than those who are only core developers in one area. We also find the core developers in both technical and social areas to have more commits with more structural changes (not just localization or bug fixes) than developers who are only core in the cochange network. For the issue data, we find the core developers in both technical and social areas to have more important tasks in the issues, e.g. approving PR and new versions or reviewing PR. For core developers who are only core in issue data, we find them to do more support work, e.g. consulting on issues or testing. We find the different classification metrics to give us the same cochange core group and similar core groups from both cochange and issue data. The classification metrics differ greatly in the issue core developer group, which interestingly does not affect the other findings.
In addition to these analyses, we investigated the group of core developers who are classified as core in cochange and in issue data. Within this combined core group, we identified five distinct developer groups, each with its unique roles and competencies. This research involved applying two indices of the different areas to get distinct activity patterns within the core group. By applying these methods, we uncovered nuanced patterns of collaboration, contribution diversity, and temporal engagement, providing a detailed understanding of the diverse roles played by core developers in both the cochange and issue areas. The time-based approach also reveals the evolving nature of these roles over time, providing a

dynamic perspective on core developer roles and dynamics.

Our results highlight the understanding of the different core developer roles, competencies, and their active involvement and impact in OSS projects. As the landscape of OSS development continues to evolve, our research contributes to a more nuanced understanding of the collaborative forces that shape these projects.

6.2 Future Work

Our research lays the foundation for several promising future research opportunities in the area of OSS development. The dynamic patterns and nuanced roles identified among core developers provide a starting point for further research.
Our time-based analysis provides a general overview of core developer roles across various projects. A deeper examination of these roles could reveal patterns influenced by project lifecycles, external events, or technological advances. Understanding how roles adapt and change over time could improve our understanding of the dynamics of OSS projects. In addition, it can be analyzed whether there are more interesting patterns in the transition between the subgroups when all developer groups are included and not just the core developers of both areas.
Future research that links the observed developer role dynamics to project outcomes such as code quality, release frequency, and community health could provide insights into the factors that contribute to the success or challenges of OSS projects.
Furthermore, by extending our investigation to different domains or specific types of projects (e.g., language-specific projects, framework-centric projects), domain-specific patterns may be uncovered. Comparative studies may reveal how different contexts influence the roles and contributions of developers within OSS communities. Although not explicitly explored in this study, future research could also explore industry-specific trends in developer distribution.

Appendix

A.1 Keywords

A.1.1 Contribution Index

Table A.1: Keywords and Weights for Contribution Index Calculation

Keyword	Description	Weight
'feat'	New feature	10
'add'	Adding functionality	5
'upgrade'	Upgrading functionality	4
'update'	Updating functionality	4
'fix'	Fixing an issue	3
'error'	Handling errors	3
'support'	Adding support	3
'config'	Configuring something	3
'merge'	Merging changes	2
'move'	Moving code/files	2
'rename'	Renaming code/files	2
'improve'	Improving code/function	2
'refactor'	Refactoring code	2
'test'	Adding or modifying tests	2
'remove'	Removing code or files	2

a.1.2 Interaction Index

Table A.2: Interaction Types and Weights for Interaction Index Calculation

Interaction Type	**Description**	**Weight**
'reviewed'	Code review	10
'merged'	Merging changes	4
'added_to_project'	Adding to a project	3
'state_updated'	Updates related to the state	3
'commented'	Various types of comments and updates	3
'assigned'	Assignment of tasks	2
'type_updated'	Updates related to the type	2
'labeled'	Adding labels to issues or pull requests	2
'review_dismissed'	Dismissing a code review	2
'head_ref_force_pushed'	Forcing a push to the branch	2
'milestoned'	Associating with a milestone	2
'commit_added'	Adding a commit	0

Bibliography

[1] Guilherme Avelino, Leonardo Passosc, Andre Horaa, and Marco Tulio Valentea. "Measuring and analyzing code authorship in 1 +1180pen source projects." In: *Science of Computer Programming* 176 (2019), pp. 14–32.

[2] Salem Bahamdain. "Open Source Software (OSS) Quality Assurance: A Survey Paper." In: *Procedia Computer Science* 56 (2015), pp. 459–464.

[3] Anand Bihari and Manoj Kumar Pandia. "Eigenvector centrality and its application in research professionals' relationship network." In: *2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, ABLAZE 2015*. IEEE, 2015, 510—514.

[4] Thomas Bock, Nils Alznauer, Mitchell Joblin, and Sven Apel. "Automatic Core-Developer Identification on GitHub: A Validation Study." In: *ACM Transactions on Software Engineering and Methodology*. ACM, 2023.

[5] Thomas Bock, Angelika Schmid, and Sven Apel. "Measuring and Modeling Group Dynamics in Open-Source Software Development: A Tensor Decomposition Approach." In: *ACM Transactions on Software Engineering and Methodology*. ACM, 2021, pp. 1–50.

[6] Phillip Bonacich and Paulette Lloyd. "Eigenvector-like measures of centrality for asymmetric relations." In: *Social networks* 23.3 (2001), pp. 191–201.

[7] Stephen P. Borgatti and Martin G. Everett. "Models of core/periphery structures." In: *Social Networks* 21.4 (2000), pp. 375–395.

[8] Amiangshu Bosu and Jeffrey C. Carver. "Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation." In: *ESEM '14: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, 1—10.

[9] Manuel Hoffmann, Frank Nagle, and Yanuo Zhou. "The Value of Open Source Software." In: *Harvard Business School Strategy Unit Working Paper No. 24-038*. SSRN, 2024.

[10] Andrejs Jermakovics, Alberto Sillitti, and Giancarlo Succi. "Mining and Visualizing Developer Networks from Version Control Systems." In: *CHASE'11: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 2011, 24—31.

[11] Mitchell Joblin. "Structural and Evolutionary Analysis of Developer Networks." Doktorarbeit. Germany: University of Passau, 2017.

[12] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. "Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics." In: *ICSE'17: Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 2017, 164—174.

[13] Mitchell Joblin, Barbara Eckl, Thomas Bock, Angelika Schmid, Janet Siegmund, and Sven Apel. "Hierarchical and Hybrid Organizational Structures in Open-Source Software Projects: A Longitudinal Study." In: *ACM Trans. Softw. Eng. Methodol.* 32.4 (2023). ISSN: 1049-331X.

[14] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. "From Developer Networks to Verified Communities: A Fine-Grained Approach." In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 2015, pp. 563–573.

[15] Luis López-Fernández, Gregorio Robles, Jesus Gonzalez-Barahona, and Israel Herraiz. "Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects." In: *International Journal of Information Technology and Web Engineering (IJITWE), 1(3)*. Continuous Volume, 2006, 27—48.

[16] Andrew Meneely, Mackenzie Corcoran, and Laurie Williams. "Improving Developer Activity Metrics with Issue Tracking Annotations." In: *WETSoM '10: Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. ACM, 2010, 75—80.

[17] Andrew Meneely and Laurie Williams. "On the Use of Issue Tracking Annotations for Improving Developer Activity Metrics." In: *Advances in Software Engineering* 2010.Article ID 273080 (2010).

[18] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. "Evolution Patterns of Open-Source Software Systems and Communities." In: *Proceedings of the International Workshop on Principles of Software Evolution*. Association for Computing Machinery, 2002, 76–85.

[19] Gustavo A. Oliva, Francisco W. Santana, Kleverton C. M. de Oliveira, Cleidson R. B. de Souza, and Marco A. Gerosa. "Characterizing Key Developers: A Case Study with Apache Ant." In: *CRIWG 2012: Collaboration and Technology*. Springer, 2012, 97—112.

[20] Gustavo Ansaldi Oliva, José Teodoro da Silva, and Marco Aurélio Gerosa. "Evolving the System's Core: A Case Study on the Identification and Characterization of Key Developers in Apache Ant." In: *Computing and Informatics* 34.3 (2015), pp. 678–724.

[21] Erzsébet Ravasz and Albert-László Barabási. "Hierarchical organization in complex networks." In: *Physical review E* 67 (2 2003), p. 026112.

[22] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Israel Herraiz. "Evolution of the core team of developers in libre software projects." In: *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 2009, pp. 167–170.

[23] Leo Spizzirri. *Justification and application of eigenvector centrality*. 2011.

[24] Maarten van Steen. *An Introduction to Graph Theory and Complex Networks*. 2010.

[25] Antonio Terceiro, Luiz Romário Rios, and Christina Chavez. "An Empirical Study on the Structural Complexity Introduced by Core and Peripheral Developers in Free Software Projects." In: *Brazilian Symposium on Software Engineering (SBES)*. IEEE, 2010, pp. 21–29.

[26] Thomas Erlebach Ulrik Brandes. *Network Analysis: Methodological Foundations*. Springer, 2005.

[27] Jin Xu, Yongqin Gao, Scott Christley, and Gregory Madey. "A Topological Analysis of the Open Souce Software Development Community." In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. 2005, 198a–198a.

[28] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi. "Revisiting the Applicability of the Pareto Principle to Core Development Teams in Open Source Software Projects." In: *IWPSE 2015: Proceedings of the 14th International Workshop on Principles of Software Evolution*. ACM, 2015, 46—55.